The Stellar Consensus Protocol (SCP)

draft-mazieres-dinrg-scp-05

Nicolas Barry, Giuliano Losa, David Mazières, Jed McCaleb, Stanislas Polu

IETF103

Tuesday, November 8, 2018

Motivation: Internet-level consensus

Atomically transact across incompatible/distrustful systems

- E.g., Transfer domain name in exchange for payment

Irrevocably delegate identifiers

- E.g., certify email user public key w/o ability to equivocate

Verify public disclosure & timestamp of information

- Build IoT device that only upgrades to public firmware

All of these can be addressed w. public append-only log

Slice infrastructures

A *slice infrastructure* is a set of *nodes* that select *quorum slices*

Each node picks quorum slices it believes speaks for the Internet

- E.g., I pick {Stanford, IETF}, you pick {Baidu, Wechat, Alibaba}
- Alibaba and Stanford both include Google in their quorum slices
- Transitively, we both depend on Google
- Want guaranteed agreement so long as Google honest

For fault tolerance, pick multiple quorum slices

- E.g., 4/5 FAANG companies, or 3/4 of servers from each FAANG

Define quorums as the transitive closure of slices

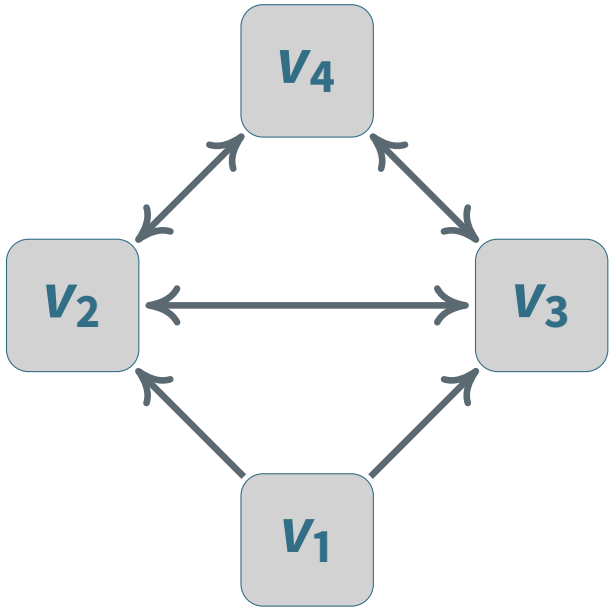
- Let $\mathbf{V}$ be all nodes, $\mathbf{Q}(v)$ be all of node v's quorum slices

Definition (Quorum)

A quorum $U \subseteq \mathbf{V}$ is a set of nodes that contains at least one slice of each of its members: $\forall v \in U, \exists q \in \mathbf{Q}(v)$ such that $q \subseteq U$

Definition (Quorum)

A quorum $U \subseteq \mathbf{V}$ is a set of nodes that encompasses at least one slice of each of its members: $\forall v \in U, \exists q \in \mathbf{Q}(v)$ such that $q \subseteq U$



$$\mathbf{Q}(v_1) = \{\{v_1, v_2, v_3\}\}$$
$$\mathbf{Q}(v_2) = \mathbf{Q}(v_3) = \mathbf{Q}(v_4) = \{\{v_2, v_3, v_4\}\}$$

Visualize quorum slice dependencies with arrows

v_2, v_3, v_4 **is a quorum—contains a slice of each member**

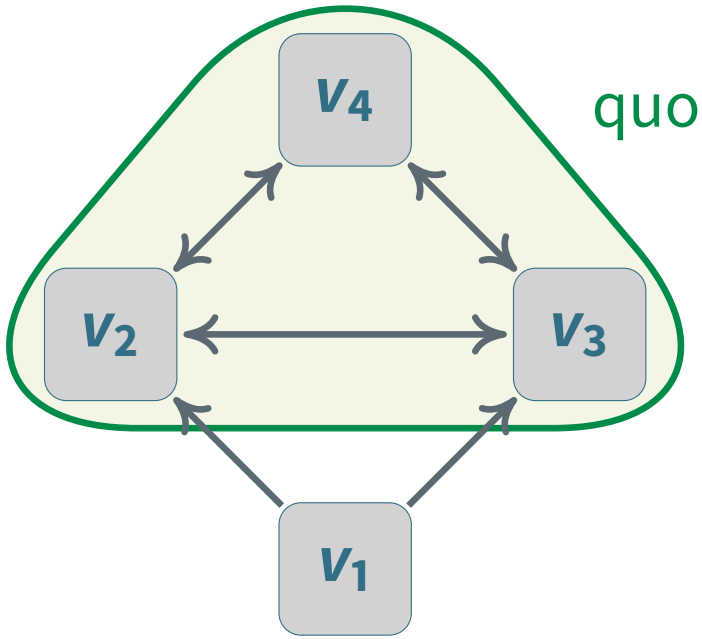
v_1, v_2, v_3 **is a slice for** v_1**, but not a quorum**

- Doesn't contain a slice for v_2, v_3, who demand v_4's agreement

$v_1, \ldots, v_4$ **is the smallest quorum containing** v_1

Definition (Quorum)

A quorum $U \subseteq \mathbf{V}$ is a set of nodes that encompasses at least one slice of each of its members: $\forall v \in U, \exists q \in \mathbf{Q}(v)$ such that $q \subseteq U$



quorum for v_2, v_3, v_4

$$\mathbf{Q}(v_1) = \{\{v_1, v_2, v_3\}\}$$
$$\mathbf{Q}(v_2) = \mathbf{Q}(v_3) = \mathbf{Q}(v_4) = \{\{v_2, v_3, v_4\}\}$$

Visualize quorum slice dependencies with arrows

v_2, v_3, v_4 **is a quorum—contains a slice of each member**

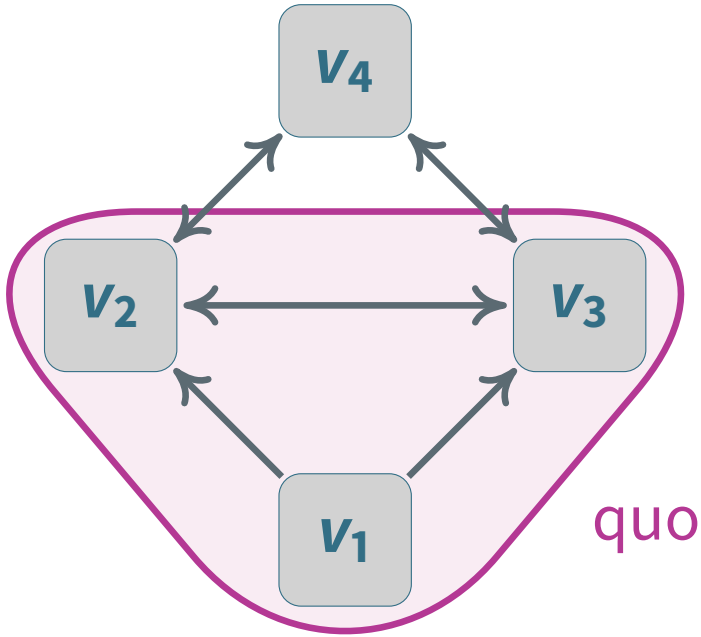
v_1, v_2, v_3 **is a slice for** v_1**, but not a quorum**

- Doesn't contain a slice for v_2, v_3, who demand v_4's agreement

$v_1, \ldots, v_4$ **is the smallest quorum containing** v_1

Definition (Quorum)

A quorum $U \subseteq \mathbf{V}$ is a set of nodes that encompasses at least one slice of each of its members: $\forall v \in U, \exists q \in \mathbf{Q}(v)$ such that $q \subseteq U$



$$\mathbf{Q}(v_1) = \{\{v_1, v_2, v_3\}\}$$
$$\mathbf{Q}(v_2) = \mathbf{Q}(v_3) = \mathbf{Q}(v_4) = \{\{v_2, v_3, v_4\}\}$$

quorum slice for v_1, but not a quorum

Visualize quorum slice dependencies with arrows

v_2, v_3, v_4 **is a quorum—contains a slice of each member**

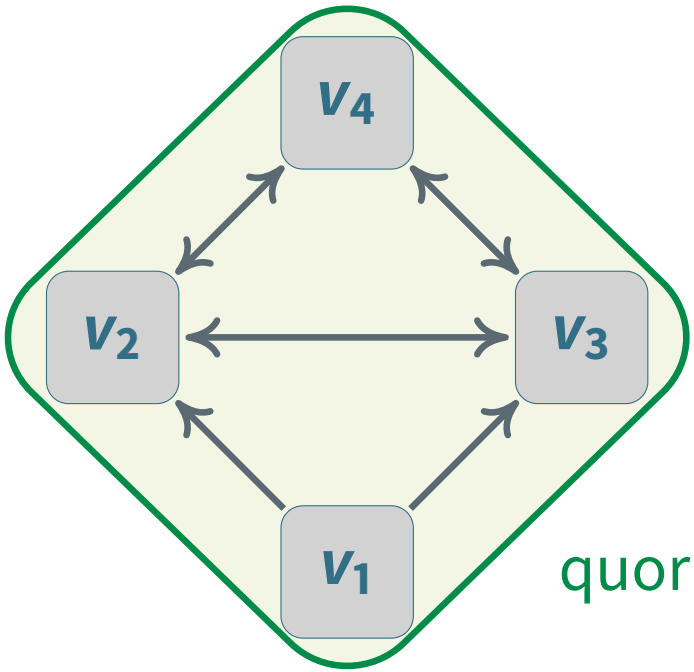
v_1, v_2, v_3 **is a slice for** v_1**, but not a quorum**

- Doesn't contain a slice for v_2, v_3, who demand v_4's agreement

$v_1, \ldots, v_4$ **is the smallest quorum containing** v_1

Definition (Quorum)

A quorum $U \subseteq \mathbf{V}$ is a set of nodes that encompasses at least one slice of each of its members: $\forall v \in U, \exists q \in \mathbf{Q}(v)$ such that $q \subseteq U$



$\mathbf{Q}(v_1) = \{\{v_1, v_2, v_3\}\}$

$\mathbf{Q}(v_2) = \mathbf{Q}(v_3) = \mathbf{Q}(v_4) = \{\{v_2, v_3, v_4\}\}$

quorum for $v_1, \ldots, v_4$

Visualize quorum slice dependencies with arrows

v_2, v_3, v_4 **is a quorum—contains a slice of each member**

v_1, v_2, v_3 **is a slice for** v_1**, but not a quorum**

- Doesn't contain a slice for v_2, v_3, who demand v_4's agreement

$v_1, \ldots, v_4$ **is the smallest quorum containing** v_1

Two important thresholds

A node *v* believes message *m* reaches...

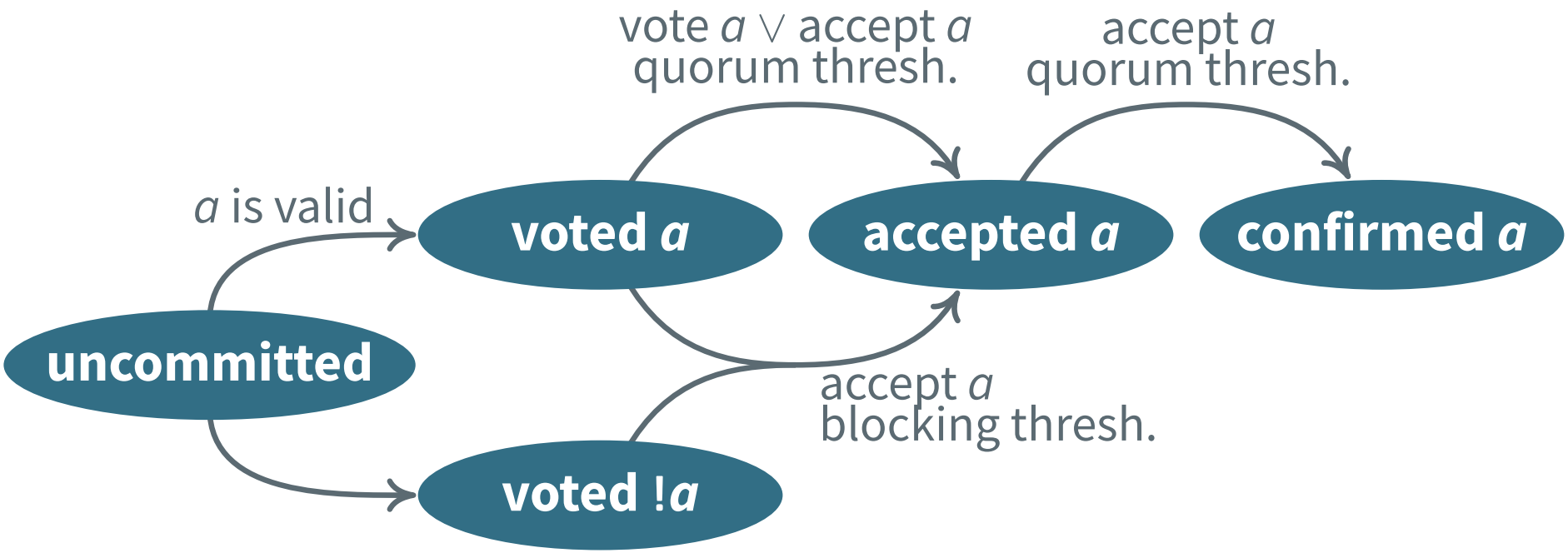
Quorum threshold – when a quorum including *v* all sends *m*

- Note *v* doesn't care about quorums it doesn't belong to (maybe Sybils)

Blocking threshold – a node sent *m* in each of *v*'s quorum slices

- Means if *v* in any honest quorum, none of its quorums can contradict *m*

Main subroutine: federated voting



Nodes vote for or against a conceptual statement *a*

- Illegal to vote for or accept two contradictory statements
- But you can vote for one statement then accept a contradictory one

When you confirm a statement, you know

- If you are *intact*, all other intact nodes will eventually confirm it
- Nodes *intertwined* with you won't confirm contradictory statements

Until you confirm a statement, it might get permanently stuck

SCP overview by phase

NOMINATE – pick some value to try to agree on

- Nodes will likely agree if network synchronous, but can disagree

PREPARE part 1 – confirm prepare(b) for ballot $\langle b.\texttt{counter}, b.\texttt{value}\rangle$

- Use federated voting to abort and commit ballots
- prepare(b) $= \{\, \text{abort}(b') \mid b' < b \ \wedge\ b'.\texttt{value} \neq b.\texttt{value} \,\}$
- b.value taken from nomination output until any ballot p is confirmed prepared, then use $p.\texttt{value}$ for highest confirmed prepared ballot p

PREPARE part 2 – accept commit(b) after confirming prepare(b)

- But if in the process you accept abort(b), go back to PREPARE part 1

COMMIT – confirm commit(b) after accepting

EXTERNALIZE – output value of confirmed committed ballot

- Also send message to optimize quorum discovery for slower nodes

Ballots details

```
struct SCPBallot {
  uint32 counter;
  Value value;
};
```

Ballots totally ordered with `counter` more significant than `value`

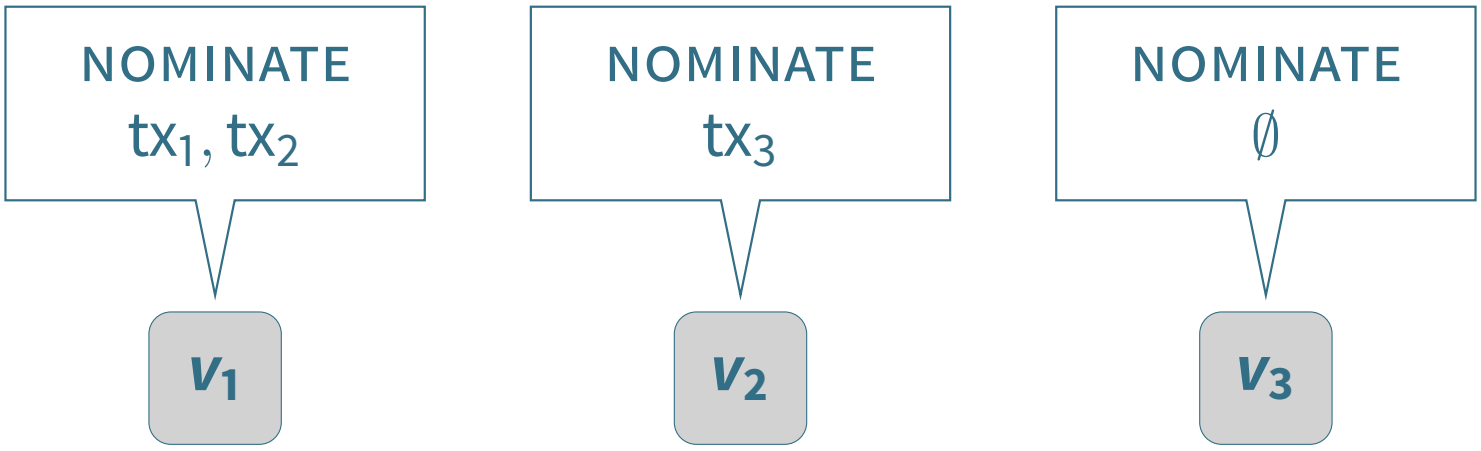
If a node confirms commit(b) for any b, it decides b.`value`

Recall prepare(b) = $\{$abort(b') $\mid b' < b \ \wedge \ b'.\texttt{value} \neq b.\texttt{value}\}$

Key invariants

- A node may vote abort(b) or commit(b) but not both (contradictory)
- A node may accept abort(b) or commit(b) but not both
- A node cannot vote commit(b) unless it first confirms prepare(b)
  $\Longrightarrow$ all committed & stuck ballots have same value

Nomination flow



Nodes nominate values and re-nominate any nominations seen

Stop adding to `votes` once any value confirmed nominated

Will converge on set of values

Deterministically combine nominations into *composite* value x

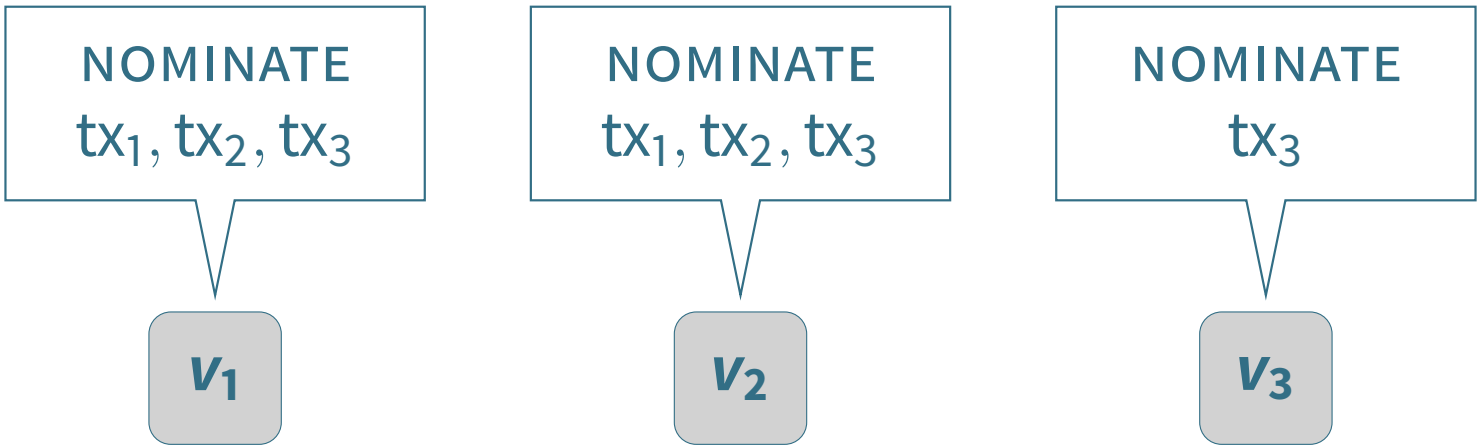
Nodes guaranteed to converge on same value x

- Complication: impossible to know when protocol has converged [FLP]

NOMINATE overlaps PREPARE to continue in background

- Ends when ballot confirmed prepared, as all intact nodes will confirm prepared ballot and use its value

Nomination flow



Nodes nominate values and re-nominate any nominations seen

Stop adding to `votes` once any value confirmed nominated

Will converge on set of values

Deterministically combine nominations into *composite* value x

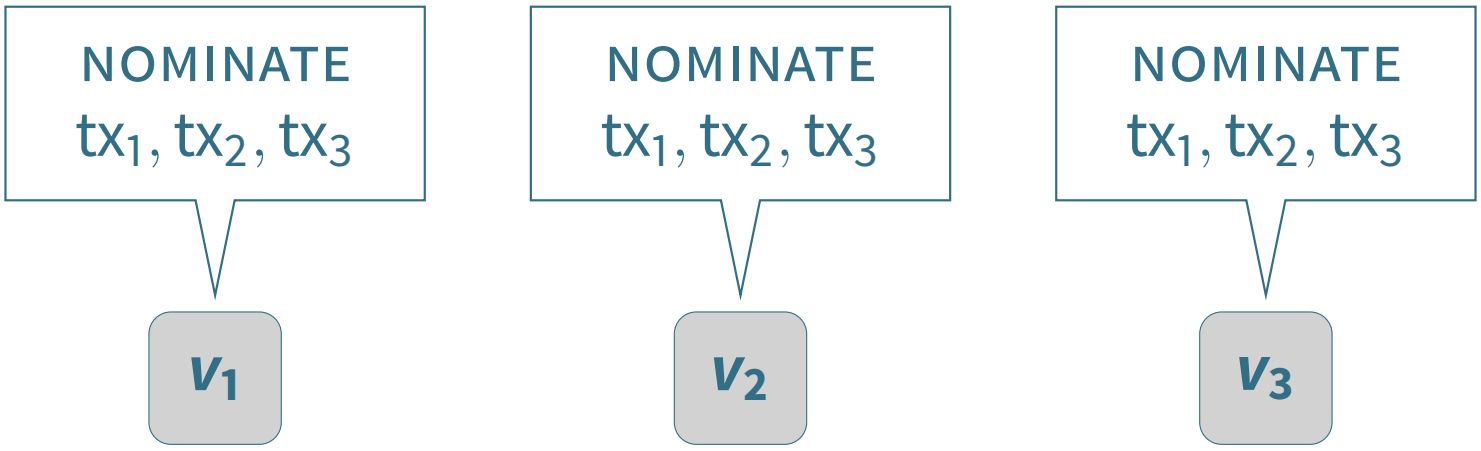
Nodes guaranteed to converge on same value x

- Complication: impossible to know when protocol has converged [FLP]

NOMINATE overlaps PREPARE to continue in background

- Ends when ballot confirmed prepared, as all intact nodes will confirm prepared ballot and use its value

Nomination flow



Nodes nominate values and re-nominate any nominations seen

Stop adding to `votes` once any value confirmed nominated

Will converge on set of values

Deterministically combine nominations into *composite* value x

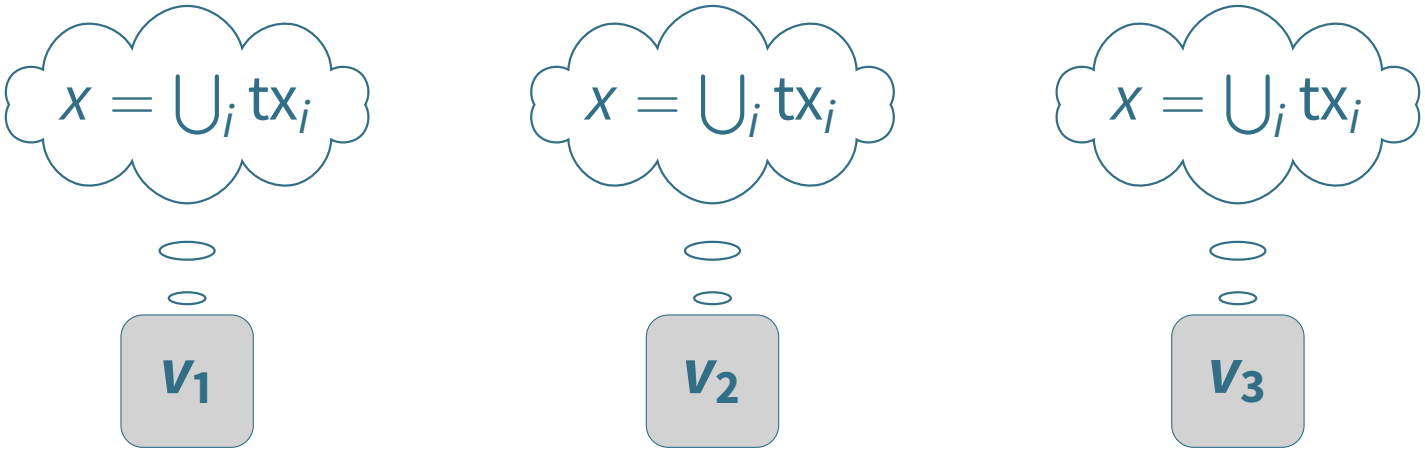
Nodes guaranteed to converge on same value x

- Complication: impossible to know when protocol has converged [FLP]

NOMINATE overlaps PREPARE to continue in background

- Ends when ballot confirmed prepared, as all intact nodes will confirm prepared ballot and use its value

Nomination flow



Nodes nominate values and re-nominate any nominations seen

Stop adding to `votes` once any value confirmed nominated

Will converge on set of values

Deterministically combine nominations into *composite* value x

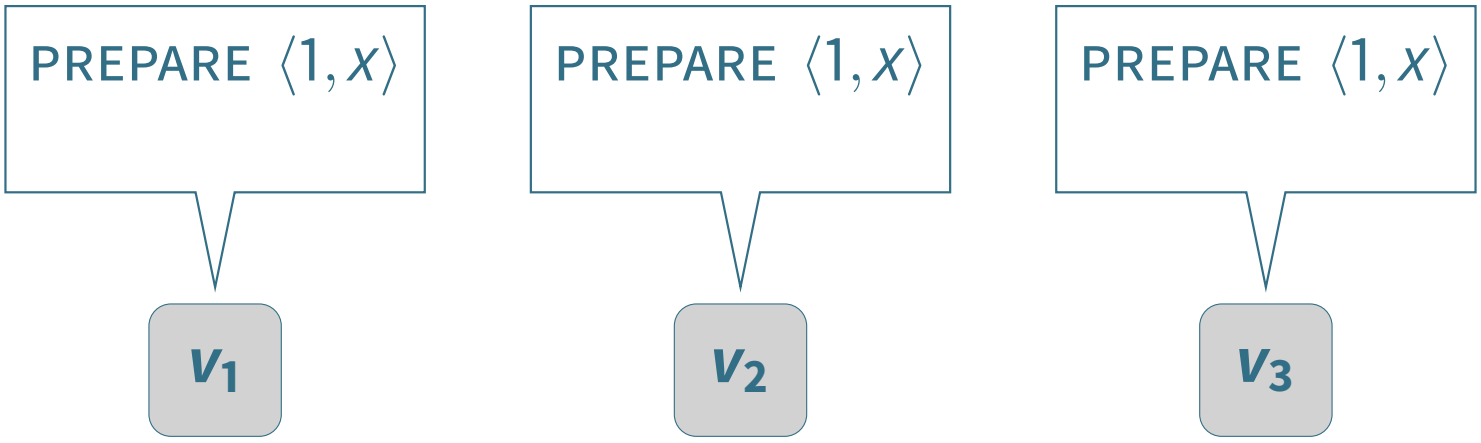
Nodes guaranteed to converge on same value x

- Complication: impossible to know when protocol has converged [FLP]

NOMINATE overlaps PREPARE to continue in background

- Ends when ballot confirmed prepared, as all intact nodes will confirm prepared ballot and use its value

Balloting flow



In the common case, will prepare and commit nominated value

Else, arm timer when ballot counter reaches quorum threshold

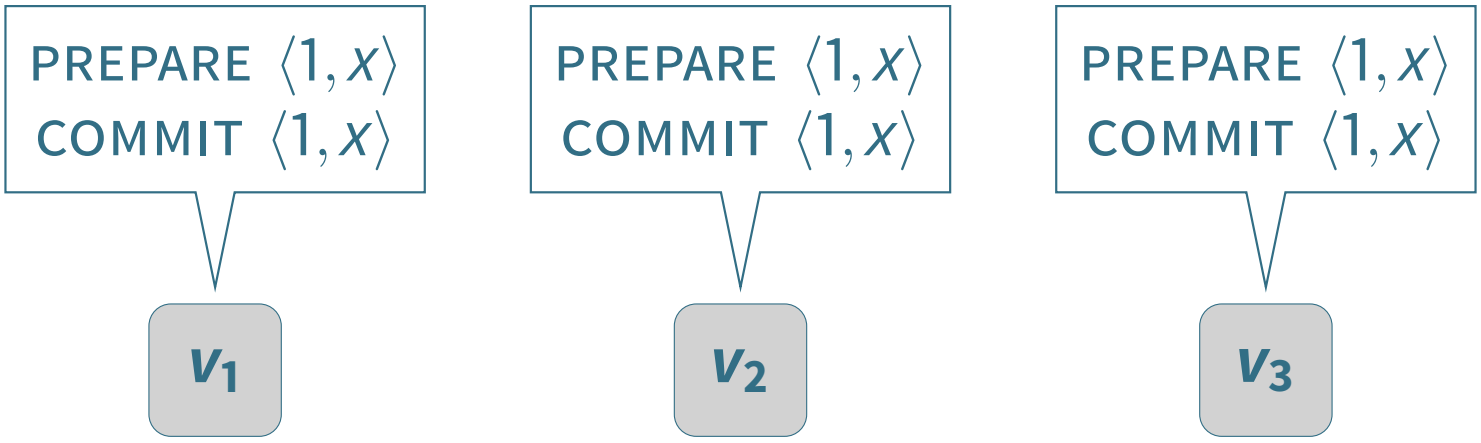
Bump counter and restart with new ballot whenever

- Timer fires
- A blocking threshold is at a higher ballot counter

Timeout lengthens as counter increases

- Intact nodes spend longer and long on same counter together
- Eventually emulates a synchronous system

Balloting flow



In the common case, will prepare and commit nominated value

Else, arm timer when ballot counter reaches quorum threshold

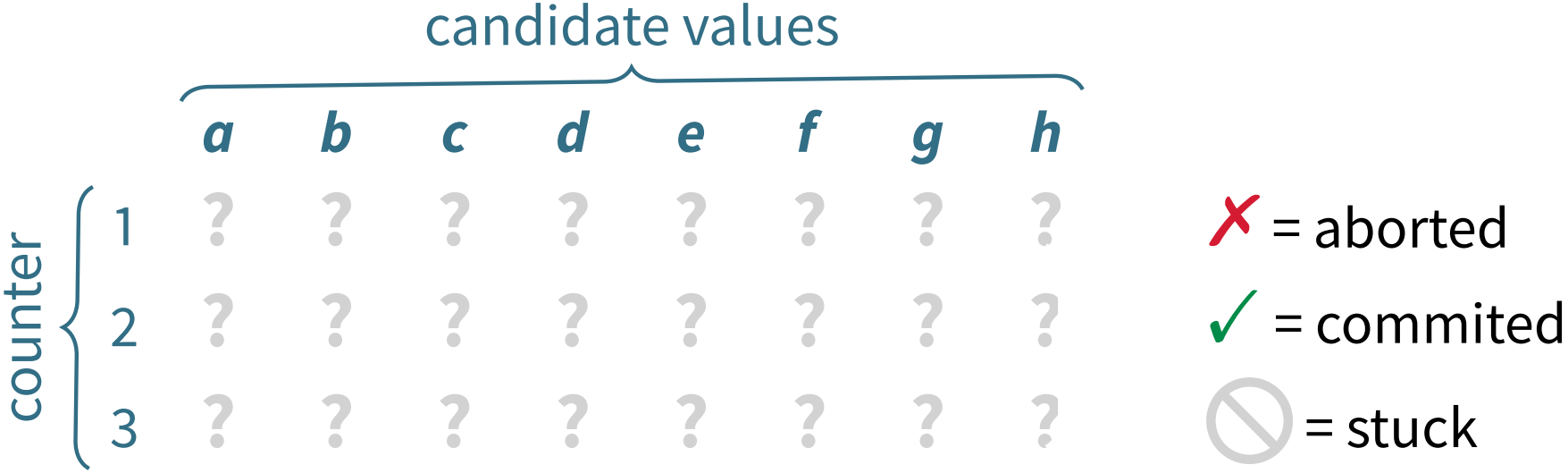
Bump counter and restart with new ballot whenever

- Timer fires
- A blocking threshold is at a higher ballot counter

Timeout lengthens as counter increases

- Intact nodes spend longer and long on same counter together
- Eventually emulates a synchronous system

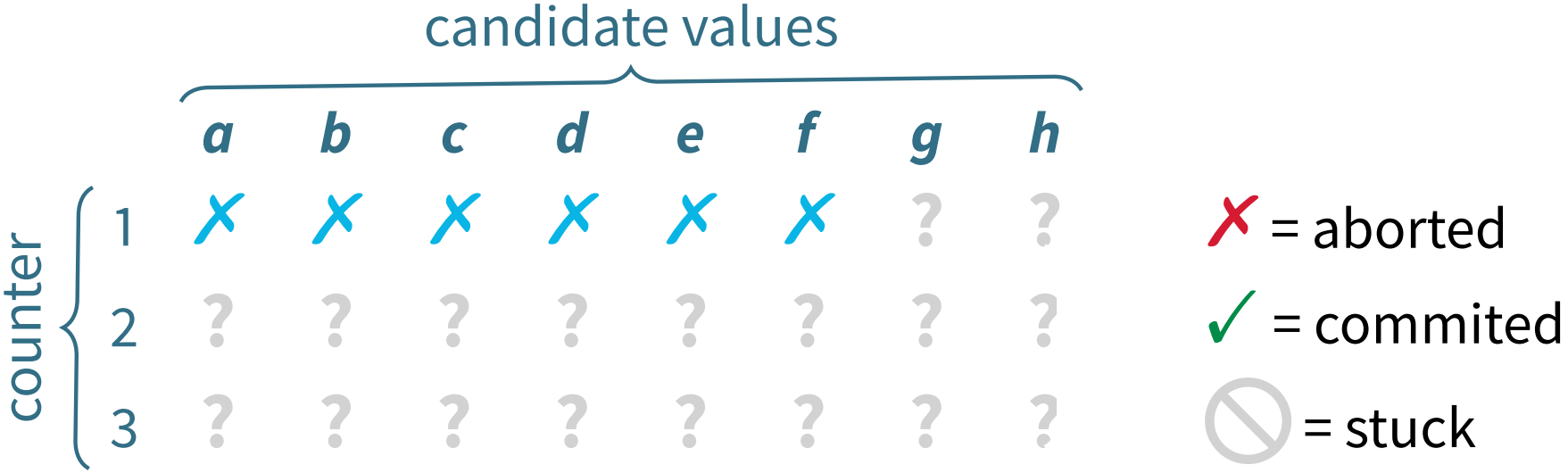
Balloting example



0. **Initially, all ballots are bivalent**
1. **Prepare $\langle 1, g \rangle$ and vote to commit it**
2. **Lose vote on $\langle 1, g \rangle$; agree $\langle 2, f \rangle$ prepared and vote to commit it**
3. **$\langle 2, f \rangle$ is stuck, so agree $\langle 3, f \rangle$ prepared and vote to commit it**
4. **Confirm commit $\langle 3, f \rangle$ and externalize f**
   - At this point nobody cares that $\langle 2, f \rangle$ is stuck

Notice how all committed & stuck ballots have same value

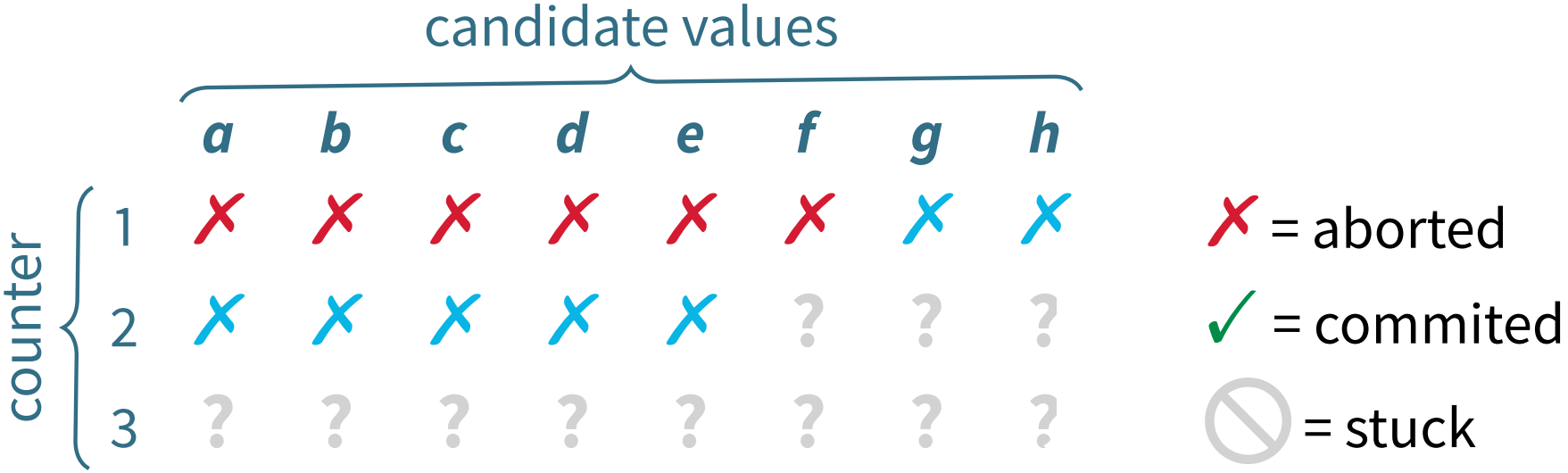
Balloting example



0. **Initially, all ballots are bivalent**
1. **Prepare $\langle 1, g \rangle$ and vote to commit it**
2. **Lose vote on $\langle 1, g \rangle$; agree $\langle 2, f \rangle$ prepared and vote to commit it**
3. **$\langle 2, f \rangle$ is stuck, so agree $\langle 3, f \rangle$ prepared and vote to commit it**
4. **Confirm commit $\langle 3, f \rangle$ and externalize f**
   - At this point nobody cares that $\langle 2, f \rangle$ is stuck

Notice how all committed & stuck ballots have same value

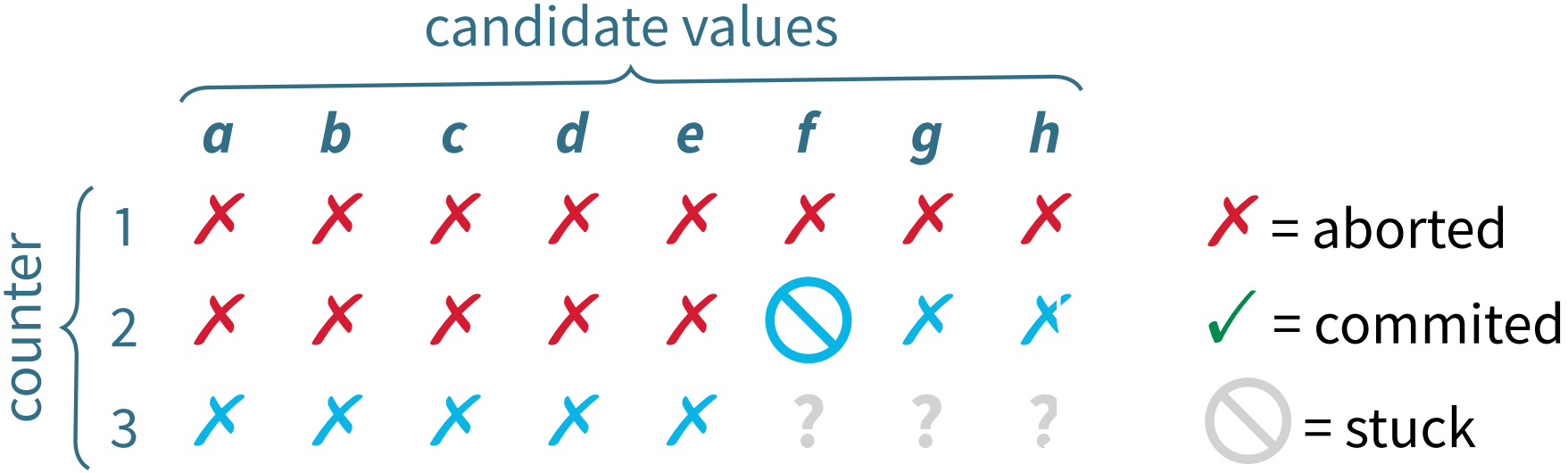
Balloting example



0. **Initially, all ballots are bivalent**
1. **Prepare $\langle 1, g \rangle$ and vote to commit it**
2. **Lose vote on $\langle 1, g \rangle$; agree $\langle 2, f \rangle$ prepared and vote to commit it**
3. **$\langle 2, f \rangle$ is stuck, so agree $\langle 3, f \rangle$ prepared and vote to commit it**
4. **Confirm commit $\langle 3, f \rangle$ and externalize f**
   - At this point nobody cares that $\langle 2, f \rangle$ is stuck

Notice how all committed & stuck ballots have same value

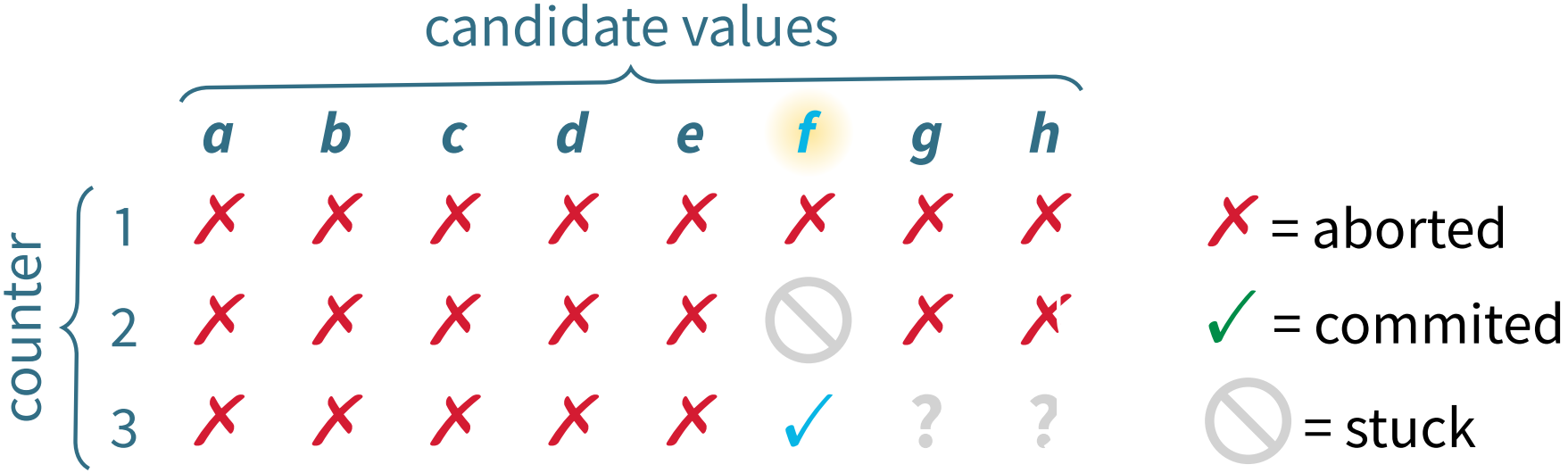
Balloting example



0. **Initially, all ballots are bivalent**
1. **Prepare $\langle 1, g \rangle$ and vote to commit it**
2. **Lose vote on $\langle 1, g \rangle$; agree $\langle 2, f \rangle$ prepared and vote to commit it**
3. **$\langle 2, f \rangle$ is stuck, so agree $\langle 3, f \rangle$ prepared and vote to commit it**
4. **Confirm commit $\langle 3, f \rangle$ and externalize f**
   - At this point nobody cares that $\langle 2, f \rangle$ is stuck

Notice how all committed & stuck ballots have same value

Balloting example



0. **Initially, all ballots are bivalent**
1. **Prepare $\langle 1, g \rangle$ and vote to commit it**
2. **Lose vote on $\langle 1, g \rangle$; agree $\langle 2, f \rangle$ prepared and vote to commit it**
3. **$\langle 2, f \rangle$ is stuck, so agree $\langle 3, f \rangle$ prepared and vote to commit it**
4. **Confirm commit $\langle 3, f \rangle$ and externalize f**
   - At this point nobody cares that $\langle 2, f \rangle$ is stuck

Notice how all committed & stuck ballots have same value

SCP prepare message (changed)

```
struct SCPPrepare {
  SCPBallot ballot;
  SCPBallot *prepared;
  uint32 aCounter;     // new -- replaces preparedPrime
  uint32 hCounter;
  uint32 cCounter;
};
```

vote-or-accept prepare(`ballot`)

if `prepared` $\neq$ **NULL: accept prepare**(`*prepared`)

accept $\{$ **abort**(b) $\mid$ b.`counter` $<$ `aCounter` $\}$

if hCounter $\neq 0$**: confirm prepare**($\langle$**hCounter**, **ballot.value**$\rangle$)

if cCounter $\neq 0$**:**
vote $\{$**commit**($\langle n,$ **ballot.value**$\rangle$) $\mid$ **cCounter** $\leq n \leq$ **hCounter**$\}$

Progress to COMMIT phase upon accepting commit(b) **for any** b

Setting the prepare fields

`ballot.counter` starts at 1, increases with timeouts/blocking sets

`ballot.value` b.`value` from highest confirmed prepare(b) (if any), else composite nomination value (if any), else b.`value` from highest accepted prepare(b) (if any), otherwise don't send `SCPPrepare` yet

`prepared` highest b for which sender accepted prepared(b)

`aCounter` `counter` (or `counter` +1) of highest accepted prepared ballot with different value from `prepared.value`

`hCounter` h.counter from highest h with confirmed prepared(h) and b.`value` $==$ h.`value` (new), else 0

`cCounter` 0 if `hCounter` $==$ 0 or internal "commit ballot" c $==$ `NULL`. Else, c.counter. Note $c \leftarrow$ `ballot` when confirmed prepared and `NULL` when accepted aborted.

Status

The good news

- Draft is stabilizing (one open question: max nomination message size)
- Existing protocol has slightly better liveness than previously proven
- At least 4 implementations: stellar-core, Bob Glickstein, Mobilecoin, Peirs Poslesland

The bad news – might want huge changes / other documents

- No hope of interoperability because no multicast specification
- Maybe we can improve liveness by re-running nomination between counters (would terminate with prob. 1 even with Byzantine nodes)
- Maybe simpler protocol for slice infrastructures (would require alternate competing draft, slow everything down)

Questions?

SCP nomination message

```
typedef opaque Value<>;

struct SCPNominate {
  Value voted<>;     // vote to nominate these values
  Value accepted<>;  // assert that these are accepted
};

union SCPStatement switch (SCPStatementType type) {
  case SCP_ST_NOMINATE:
    SCPNominate nominate;
  /* ... */
};
```

Nodes broadcast nominated values in `voted`

- Initially vote values in all received votes (ignoring optimization here)

Upon accepting nomination of a**, move from** `voted` **to** `accepted`

Stop voting for new values once any is confirmed nominated

- But continue accepting and repeating votes already cast

Stop sending `SCPNominate` **when ballot confirmed prepared**

- Means NOMINATION phase overlaps with PREPARE phase

SCP commit message

```
struct SCPCommit {
    SCPBallot ballot;
    uint32 preparedCounter;
    uint32 hCounter;
    uint32 cCounter;
};
```

$\{$**accept commit**$(\langle n, \texttt{ballot.value}\rangle) \mid \texttt{cCounter} \leq n \leq \texttt{hCounter}\}$

vote-or-accept prepare$(\langle \infty, \texttt{ballot.value}\rangle)$

accept prepare$(\langle \texttt{preparedCounter}, \texttt{ballot.value}\rangle)$

confirm prepare$(\langle \texttt{hCounter}, \texttt{ballot.value}\rangle)$

$\{$**vote commit**$(\langle n, \texttt{ballot.value}\rangle) \mid n \geq \texttt{cCounter}\}$

SCP externalize message

```
struct SCPExternalize {
    SCPBallot commit;
    uint32 hCounter;
};
```

$\{\textbf{accept commit}(\langle n, \texttt{commit.value}\rangle) \mid n \geq \texttt{commit.counter}\}$

$\{\textbf{confirm commit}(\langle n, \texttt{commit.value}\rangle) \mid \texttt{commit.counter} \leq n \leq \texttt{hCounter}\}$

$\textbf{accept prepare}(\langle \infty, \texttt{commit.value}\rangle)$

$\textbf{confirm prepare}(\langle \texttt{hCounter}, \texttt{commit.value}\rangle)$

By the time you send this, already externalized `commit.value`

- Means you have confirmed committed a ballot with `commit.`*value*
- Goal is definitive record to help other nodes prove value/catch up